Authentication in PHP: Part 2

Using Auth.php to build a Secure App

Using the Auth.php

Using the Auth.php, we can implement authentication features easily.

- login
- logout
- register
- change password
- update profile
- dashboard

login.php

Client-side code

- When users access login.php, the PHP server runs the script and returns the generated HTML code to the client's browser.
    - All the <?= ... => part will be replaced with values by the PHP server.
- This is a way to make a POST request with the user name and password.

```
    <div class="form-container">
        <form method="post">
            <div class="form-group">
                <label><strong>Username:</strong></label>
                <input type="text" name="username"
                       value="<?= htmlspecialchars($_POST['username'] ?? '') ?>"
                required>
            </div>

            <div class="form-group">
                <label><strong>Password:</strong></label>
                <input type="password" name="password" required>
            </div>

            <button type="submit">🔑 Login</button>
        </form>
    </div>

    <div class="links">
        <p><a href="register.php">Don't have an account? Register here</a></p>
        <p><a href="index.php">← Back to main menu</a></p>
    </div>
```

Server-side code

- Users click the "login" button.
- If `action` is not set, submitting the form will send the data to the current page's URL by default. This means the browser will POST the form data to the same page the form is on
- PHP (login.php) receives the POST request.

- Check the session.
- If the user is already logged in, direct to the page `dashboard.php`.

```php
<?php
require_once 'Auth.php';
require_once 'SessionAuth.php';

$auth = new Auth();
$session = new SessionAuth();

// Check if already logged in
if ($session->is_logged_in()) {
    header('Location: dashboard.php');
    exit;
}

$error_message = '';
$timeout_message = '';

// Check for session timeout
if (isset($_GET['timeout'])) {
    $timeout_message = 'Your session has expired. Please login again.';
}
```

- If not, check the method is POST.
- Then, use the Auth class function to check the user name and password.
- Redirect to the dashboard.php.

```php
if ($_POST) {
    try {
        // Get form data
        $username = trim($_POST['username'] ?? '');
        $password = $_POST['password'] ?? '';

        // Basic validation
        if (empty($username) || empty($password)) { throw new Exception('Username and password are required'); }
        // Attempt login with Auth class
        $user = $auth->login($username, $password);
        // Login successful - create session
        $session->login_user($user);
        // Redirect to dashboard
        header('Location: dashboard.php');
        exit;

    } catch (Exception $e) {
        $error_message = $e->getMessage();
    }
}
?>
```

register.php

Client-side code

- Client chooses username, password, email, and confirmation of password.

```
        <form method="post">
            <div class="form-group">
                <label><strong>Username:</strong> *</label>
                <input type="text" name="username"
                       value="<?= htmlspecialchars($_POST['username'] ?? '') ?>"
                required>
            </div>

...

            <div class="form-group">
                <label><strong>Confirm Password:</strong> *</label>
                <input type="password" name="confirm_password" required>
            </div>

            <button type="submit">🚀 Create Account</button>
        </form>
```

Server-side code

- The server receives the user's POST request.
- The server validates information.
- The server checks if the email and confirm_email are the same.
- When everything is OK, the user is registered.

```php
if ($_POST) {
    try {
        // Get form data
        $username = trim($_POST['username'] ?? '');
        $email = trim($_POST['email'] ?? '');
        $password = $_POST['password'] ?? '';
        $confirm_password = $_POST['confirm_password'] ?? '';
        // Basic validation
        if (empty($username) || empty($email) || empty($password)) {
            throw new Exception('All fields are required');
        }
        if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
            throw new Exception('Invalid email format');
        }
        if ($password !== $confirm_password) {
            throw new Exception('Passwords do not match');
        }
        // Register user with Auth class
        $user = $auth->register($username, $password, $email);
        // Clear form on success
        $_POST = [];

    } catch (Exception $e) {
        $error_message = $e->getMessage();
    }
}
?>
```

change_password.php

- The server validates old, new, and confirm_password.
- Then, update the password using the Auth method.

```php
if ($_POST) {
    try {
        // Get form data
        $old_password = $_POST['old_password'] ?? '';
        $new_password = $_POST['new_password'] ?? '';
        $confirm_password = $_POST['confirm_password'] ?? '';

        // Basic validation
        if (empty($old_password) || empty($new_password) || empty($confirm_password)) {
            throw new Exception('All fields are required');
        }
        if ($new_password !== $confirm_password) { throw new Exception('New passwords do not match'); }
        if ($old_password === $new_password) { throw new Exception('New password must be different from current password'); }
        // Change password using Auth class
        $auth->change_password($current_user['id'], $old_password, $new_password);
        // Clear form on success
        $_POST = [];
    } catch (Exception $e) { ... }
}
```

update_profile.php (Change email)

- This script is for a secure user profile update handler, specifically for updating a logged-in user’s email address.
- It enforces authentication, provides solid validation, and protects sensitive user data from malicious or unintended changes — all of which are essential for a well-designed, secure web app.
    - Only logged-in users can change their data.
    - Only the email field is updatable.

- The update_profile() method of the Auth class makes sure that only the `email` field can be updated.

```
    public function update_profile($user_id, $updates) {
        // Only allow safe fields to be updated
        $allowed_fields = ['email'];
        $safe_updates = [];

        foreach ($allowed_fields as $field) {
            if (isset($updates[$field])) {
                // Special validation for email
                if ($field === 'email') {
```

```php
if ($_POST) {
    try {
        // Get form data
        $email = trim($_POST['email'] ?? '');

        // Basic validation
        if (empty($email)) {
            throw new Exception('Email is required');
        }

        if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
            throw new Exception('Invalid email format');
        }

        // Update profile using Auth class
        $updated_user = $auth->update_profile($current_user['id'], [
            'email' => $email
        ]);

        $success_message = 'Profile updated successfully!';

        // Refresh user details
        $user_details = $auth->find_user_by_id($current_user['id']);

    } catch (Exception $e) {
        $error_message = $e->getMessage();
    }
}
```

dashboard.php

- When users are securely logged in, they are redirected to `dashboard.php`.
- It requires user authentication at the very top of your PHP file before any HTML output or protected logic is shown.

```php
require_once 'SessionAuth.php';
$session = new SessionAuth();
// Require authentication
$session->require_auth();
```

user_management.php

- This script manages users using the Auth methods, including activating and deactivating users.
    - For deactivating users, it should prevent users from deactivating themselves.

```
if ($user_id == $current_user['id'] && $action === 'deactivate') {
    throw new Exception('You cannot deactivate your own account');
}
```

```php
if ($_POST) {
    try {
        $action = $_POST['action'] ?? '';
        $user_id = $_POST['user_id'] ?? '';

        if (empty($action) || empty($user_id)) {
            throw new Exception('Invalid request parameters');
        }
        // Prevent users from deactivating themselves
        if ($user_id == $current_user['id'] && $action === 'deactivate') {
            throw new Exception('You cannot deactivate your own account');
        }
        if ($action === 'activate') {
            $auth->activate_user($user_id);
            $success_message = "User activated successfully!";
        } elseif ($action === 'deactivate') {
            $auth->deactivate_user($user_id);
            $success_message = "User deactivated successfully!";
        } else {
            throw new Exception('Invalid action specified');
        }

    } catch (Exception $e) {
        $error_message = $e->getMessage();
    }
}

// Get user statistics and all users
$user_stats = $auth->get_user_stats();
$all_users = $auth->get_all_users(true); // Include inactive users
```

logout.php

```php
$session = new SessionAuth();

// Check if user was logged in
$was_logged_in = $session->is_logged_in();
$username = $session->get_current_user()['username'] ?? 'User';

// Logout user
$session->logout();
```

Security in Web Programming

Best Practices

1. Password Security

```php
// ✅ Use strong password hashing
$hash = password_hash($password, PASSWORD_DEFAULT);

// ✅ Verify passwords properly
if (password_verify($input_password, $stored_hash)) {
    // Login successful
}

// ✅ Enforce password strength
if (strlen($password) < 8 || !preg_match('/[A-Z]/', $password)) {
    throw new Exception('Password too weak');
}
```

2. Session Security

```php
// ✅ Regenerate session ID on login
session_regenerate_id(true);

// ✅ Implement session timeout
if (time() - $_SESSION['login_time'] > 1800) { // 30 minutes
    session_destroy();
}

// ✅ Validate session data
if (!isset($_SESSION['user_id']) || !is_numeric($_SESSION['user_id'])) {
    session_destroy();
}
```

3. Input Validation

```php
// ✅ Always validate and sanitize input
$username = filter_var($_POST['username'], FILTER_SANITIZE_STRING);
$email = filter_var($_POST['email'], FILTER_VALIDATE_EMAIL);

// ✅ Use prepared statements for database queries
$stmt = $pdo->prepare("SELECT * FROM users WHERE username = ?");
$stmt->execute([$username]);
```

Common Authentication Vulnerabilities

1. SQL Injection

```
// ❌ Vulnerable
$sql = "SELECT * FROM users WHERE username = '{$_POST['username']}'";

// ✅ Safe
$stmt = $pdo->prepare("SELECT * FROM users WHERE username = ?");
$stmt->execute([$_POST['username']]);
```

2. Session Fixation

```php
// ❌ Vulnerable
session_start();
$_SESSION['user_id'] = $user['id'];

// ✅ Safe
session_start();
session_regenerate_id(true); // Regenerate session ID
$_SESSION['user_id'] = $user['id'];
```

3. Timing Attacks

```
// ❌ Vulnerable - different response times reveal info
if (!$user) {
    return "User not found";
}
if (!password_verify($password, $user['hash'])) {
    return "Wrong password";
}

// ✅ Safe - consistent response
if (!$user || !password_verify($password, $user['hash'] ?? '')) {
    return "Invalid credentials";
}
```

Key Takeaways

Authentication Fundamentals

1. **Never store plain passwords** - always use `password_hash()`
2. **Verify with `password_verify()`** - secure comparison
3. **Use sessions for state** - maintain login status
4. **Validate all input** - prevent injection attacks

Security Features

- **Rate limiting** prevents brute force attacks
- **Session regeneration** prevents fixation attacks
- **Password strength** enforces security standards
- **Input validation** blocks malicious data

Best Practices

- Separate authentication logic into classes
- Use proper error handling with try/catch
- Implement consistent error messages
- Log security events for monitoring

Real-World Applications

- User registration and login systems
- Admin panels and dashboards
- API authentication
- Multi-user applications